THE INFLUENCE OF PROGRAM DYNAMICS ON PERFORMANCE

by

R. Bassein

Memorandum No. UCB/ERL M82/33

5 May 1982

THE INFLUENCE OF PROGRAM DYNAMICS ON PERFORMANCE

by

Richard Bassein

Memorandum No. UCB/ERL M82/33

5 May 1982

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

The Influence of Program Dynamics on Performance

Richard Bassein[1]

1. Introduction

To model a program for simulation, one may construct a job script, a detailed trace in virtual time of demands on the computer system's resources [Mead78]. In general, the load imposed by these demands changes throughout the life of the program; thus the job script reflects the dynamic behavior of the program. This job script model will be called the "raw model". One may eliminate the dynamics in the model by replacing the original pattern of demands by a pattern, periodic in virtual time, which distributes the load evenly throughout the program's execution. The resulting model will be called the "periodic model". It should be noted that this description leaves undefined a certain parameter: the number of periods into which the life of a particular program should be divided. The choice of this parameter will be discussed later.

The question considered in this report is: How much is the prediction of performance altered when the raw models of the programs in a workload are reduced to periodic models? In other words: what is the influence of the dynamics of program resource demands on performance? In practical terms: what loss of accuracy may one expect if program models and simulators are simplified by making them periodic and deterministic? The program performance measures I used in evaluating the models are based on the waits each program encounters at each server. The system performance measures are the throughput and the queue lengths at the servers.

---

[1]Partially supported by NSF grant MCS80-12900 and Mills College, Oakland, CA.

I studied the above question by preparing raw job script models for programs of an interactive workload, deriving periodic models from them, running both on simulators and comparing the results. My general conclusions were that throughput was hardly influenced at all by program dynamics and that queue lengths and the program performance measures were influenced more by tampering with the distributions, rather than dynamics, of the programs' resource usages.

2. Disclaimers

Several issues are not addressed by this report as result of limitations on the time allotted to its preparation.

Perhaps the most glaring omission is the lack of validation of the raw job script model of the programs by comparison with actual program runs. This omission can be defended on two grounds. First, the comparison between the the raw job script model and the periodic one is the main focus of this report, hence the extent to which the raw model represents the original programs' performances is secondary. Also, Mead and Schwetman have validated the accuracy of a job script model in [Mead78], and their method of building job scripts was followed closely enough to justify confidence in my results.

As Mead and Schwetman point out, job scripts cannot model paging behavior, since this behavior is dependent on the total load on the system. Since I modeled a virtual memory system, I had to reach a compromise on this point. Because the system I modeled does not pre-load, but brings in pages only on demand, each program exhibits, throughout its execution, a base page fault pattern from its demand loading. I measured this pattern by running the programs on a system lightly loaded enough to allow each program its entire memory demand. The load-dependent paging pattern is added on top of this.

Thus the raw model assigns to each program a base fault rate for each cpu request and the simulator adds to this a fault rate computed from a parameter (called the "thrash factor") of the program and from the total memory demand at the time. The thrash factor is used to select a lifetime curve from an ad-hoc one-parameter family of possibilities. Ideally the thrash factor should be measured during the life of the program, and an appropriate value for it assigned to each cpu request of the raw model; since such a task would have greatly complicated the measurement phase of the project, I used a single ad-hoc value of the thrash factor for all cpu requests of all jobs. The periodic model assigns a constant base fault rate and a thrash factor, equal to that used in the raw model, to each program. The simulators' handling of page faults will be discussed in detail in the next section.

Only one mix of programs, the one actually occurring on the machine being used, was tested (although with three different numbers of users and distributions of between-job think times); the results would make a stronger statement if several different natural mixes of programs had been used. In particular, the workload available for modeling was heavily weighted toward cpu use.

It would have been interesting to compare both the raw and periodic models with stochastic and queueing models based on the data from the raw job script. I focused on deterministic simulation for two reasons. By comparing the performance of the raw job script model with that of the periodic one, I was attempting to isolate the influence of the programs' dynamics on their performance. In addition, certain applications, such as the use of modeling by an operating system to adjust itself to changes in the workload [Serazzi81], may require simulation runs too short for the appropriate use of stochastic methods: stochastic methods, including the methods of queueing theory, are directed toward the computation of steady state parameters, and these applications are concerned with short-term, transient behavior.

3. The system, the workload, and the simulators

The system I modeled is a VAX 11/780[2] running Berkeley UNIX[3] with virtual memory. The specifications for the components are:

COMPONENT	SPECIFICATIONS
main memory	4 megabytes
cpu	1,000,000 instructions/second
disks(2)	mean seek: 0.03 seconds mean latency: 0.006 seconds transfer: 806 kilobytes/second
disks(2)	mean seek: 0.03 seconds mean latency: 0.0083 seconds transfer: 1200 kilobytes/second
terminals	up to 80 (9600 baud and 300 baud)

Using data from the system's accounting program, I selected 17 programs based on number of times used, total cpu time, total real time, and total I/O activity. I measured the resource demands of each of the selected programs with a program that reads, once a second, an executing program's resource usage data that the operating system maintains; the code for the measurement tool and job script preparation programs appears in Appendix A of [Bassein82]. I prepared terminal scripts for the simulator invoking the selected programs in the approximate proportions in which they appeared in the accounting data, with between-job think times chosen from an exponential distribution with given mean (30 seconds for the workloads I tested). Descriptions of the workloads appear in Appendix I.

The two simulators, "raw" and "periodic", use the same basic system model but differ in those respects necessary for handling raw and periodic program models, as described below. The code for both simulators appears in Appendix C of [Bassein82]. The basic system model consists of a cpu, 4 disks, and terminals.

---

[2]VAX is a trademark of Digital Equipment Corporation.

[3]UNIX is a trademark of Bell Laboratories.

The service discipline at the cpu is, like that of the system being modeled [Joy81], 32-level foreground-background with a 250 millisecond quantum at each level. Incoming jobs do not pre-empt a job in execution. The jobs in level 0 have highest priority, those in level 31, lowest priority. The level number of a job in a background queue decays (to higher priority!) by a factor of 0.9 per second until that job gains access to the cpu. As will be seen in the next section, a quantum-based service discipline presents most of the difficulty in passing from the raw to the periodic model.

Both simulators use FCFS scheduling at the disks. (The simulators only use three disks since it was not possible to identify program usage of the fourth.) The raw simulator generates seek and latency times for the disks using pseudo-random numbers. The periodic simulator sends each program to each disk once in each period: the number of blocks transferred is the same each period for that program and that disk. This number, possibly fractional, is determined by dividing the total number of blocks transferred by that program at that disk during its execution by the number of periods, in order to preserve the total transfer time. The periodic simulator uses fixed seek and latency delays for each disk. However, these must be adjusted according to the number of periods into which the execution of the program is divided in order to preserve the total uniprogramming execution time of the program. Thus, if the raw model for a given program makes N visits to a certain disk and the periodic model has M periods, hence makes M visits to that disk, the seek and latency delays for that disk in the periodic model must be multiplied by N/M to compensate for the change in the number of visits.

During a cpu service, the raw simulator uses the executing job's current page fault rate and thrash factor, the current total memory demand on the system, and pseudo-random numbers to compute the time until the job's next page fault; if this lifetime is less than the duration of the current burst (as truncated

by the service discipline), then the job is sent for a page fault after executing for the lifetime. The periodic simulator determines a (not necessarily integral) number of page faults for each job, depending on the job's (fixed) page fault rate and thrash factor, the (fixed) length of the cpu request (not truncated), and the current total memory demand on the system, at the beginning of each cpu request. Again, the seek and latency times must be adjusted according to the total number of page fault requests and the number of periods.

The periodic simulator takes advantage of the periodicity of each job script by reading in only one period and then repeating its sequence of resource demands. The choice of the length of a period for each program, and thus the size of the requests during a period, will be discussed in the next section. Thus the storage requirements for the periodic models are a small fraction of those for the raw models: statically, the periodic models occupy about 2% of the space that the raw models take for the programs I modeled; dynamically, the periodic simulator uses less than one half the space that the raw one does.

Since I focused on short simulation times, and was therefore concerned with transients and non-stationary behavior, an individual run of the raw simulator could not be expected to produce statistically meaningful results. Thus the raw simulator repeats the simulation with a given workload a specified number of times, with new seeds for the pseudo-random number generators for each run. The means and standard deviations of the performance indices based on the repeated runs are reported by the simulator. Since the period simulator is completely deterministic, its results must be judged from a single run with each workload.

Both simulators start statistics collection when a given number of terminals become active for the first time and stop when fewer than that same number have yet to complete their terminal scripts.

4. Results

The output from the simulators appears, along with a description of the workloads in Appendix I.

To put the effect of the quantum-based cpu service discipline in perspective, first consider the results for FCFS scheduling at the cpu as well as at the disks, obtained by setting the quantum to a value larger than any cpu request. The period length for each periodic model was chosen to make the length of its periodic cpu request equal to the mean of the lengths of the cpu requests of the corresponding raw model. Note that values displayed for simulation with the raw model represent means based on 10 runs; throughout the entire report, 95% confidence limits for the system performance indices of the raw models correspond to maximum errors of 2% and, in most cases, less than 1% . The following table presents results for the system indices from simulation with 16 terminals:

FCFS SCHEDULING AT THE CPU: SYSTEM PERFORMANCE

PERFORMANCE INDEX	RAW	PERIODIC	%ERROR
throughput	0.0935	0.0916	-2.0
highest priority cpu queue length	3.61	3.63	+0.6
disk 1 queue length	0.183	0.177	-3.3
disk 2 queue length	0.0216	0.0212	-1.9
disk 3 queue length	0.0301	0.0288	-4.3

The stability of the system indices is not surprising in the light of the results of Denning and Buzen [Denning77].

The program indices I measured for each program were the stretch factor, that is, wait time plus service time divided by service time, at the cpu and the waits at the disks. (Wait and service times here are the totals for one execution of the program.) As a result of the actual cpu service discipline, the wait time a

job experiences varies directly with the length of its cpu bursts; thus it seems more appropriate to measure the stretch factor than just the wait there.

Throughout all the simulations, the disk waits were completely scrambled (in terms of percent change) in passing from raw models to periodic models. The effect on the rest of the results was small, however, since the workload was so heavily oriented toward the cpu that a program's total wait at all the disks never exceeded 1.5% of its total execution time. In fact, it may be the relatively light use of the disks which caused the results there to be so unpredictable. Consequently, the only program performance indices that will be considered are the stretch factors at the cpu.

Again using FCFS scheduling at the cpu, when raw models were replaced by periodic models, the mean change in the stretch factor at the cpu was 12.0%, with a maximum change for any program of 26.9%. The distribution of errors was:

FCFS SCHEDULING AT THE CPU:
ERROR IN STRETCH FACTORS FOR PERIODIC MODELS

%ERROR	NUMBER OF PROGRAMS
25 - 30	*
20 - 25	**
15 - 20	**
10 - 15	***
5 - 10	*******
0 - 5	**

Getting reasonable results, where possible, with a quantum-based service discipline at the cpu requires more analysis in choosing the period length. Choosing, as I did initially, a program's period length to make the periodic cpu request equal the mean of the raw model's cpu requests will affect the stretch factor at the cpu very seriously (although the effect on some system indices may be much less severe). For example, although the raw model of a program

may have a mean cpu request of just under a quantum, it may still have a significant portion of requests using several quanta, each such request incurring a significantly longer wait. With the above choice of period length, the periodic model will never experience these longer waits. To compensate for this problem, I adjusted period lengths according to the following analysis. Suppose that raw model jobs experience an expected wait of w_k for a cpu request exceeding k-1 quanta but not exceeding k quanta. If the fraction of a particular job's cpu requests that exceed k-1 but not k quanta is p_k, then the expected total wait experienced by the job is $w = \sum p_k w_k$. The constant request of the periodic model should then select that w_k which is closest to w. Three problems with this analysis are: when the entire workload is converted to periodic models, the w_k will change; the gaps between the values of the w_k may not allow the reasonable approximation of w by a w_k; the computation of the w_k depends on unknown performance parameters, such as the cpu utilization. (See Appendix II for the computation of the w_k and the adjustment of period lengths.) Nevertheless, a coarse correction based on these considerations yielded significant improvement in the values of the stretch factor at the cpu. In the following, "periodic" will refer to periodic models with cpu requests equal to the mean of the raw model requests and "adjusted" will refer to periodic models with period length adjusted as suggested above.

As seen in the tables below, the throughput continues to be stable while the queue lengths are very erratic.

FOREGROUND-BACKGROUND SCHEDULING AT THE CPU: SYSTEM PERFORMANCE FOR THREE LOADS

12 terminals:

INDEX	RAW	PERIODIC	%ERROR	ADJUSTED	%ERROR
throughput	0.0807	0.0780	-3.3	0.0806	-0.1
cpu queue 0	0.214	1.703	+695.8	0.993	+364.0
disk 1 queue	0.0776	0.155	+99.7	0.161	+107.5
disk 2 queue	0.0029	0.0186	+541.4	0.0205	+606.9
disk 3 queue	0.0027	0.0263	+874.1	0.0270	+900.0

16 terminals:

INDEX	RAW	PERIODIC	%ERROR	ADJUSTED	%ERROR
throughput	0.0912	0.0914	+0.2	0.0915	+0.3
cpu queue 0	1.62	1.98	+22.2	1.72	+6.2
disk 1 queue	0.185	0.184	-0.5	0.198	+7.0
disk 2 queue	0.0217	0.0222	+2.3	0.0233	+7.4
disk 3 queue	0.0305	0.0298	-2.3	0.0302	-1.0

24 terminals:

INDEX	RAW	PERIODIC	%ERROR	ADJUSTED	%ERROR
throughput	0.0994	0.1010	+1.6	0.1003	+0.9
cpu queue 0	2.45	3.32	+35.5	3.10	+26.5
disk 1 queue	0.234	0.244	+4.3	0.251	+7.3
disk 2 queue	0.0565	0.0744	+31.7	0.0691	+22.3
disk 3 queue	0.0312	0.0281	-9.9	0.0308	-1.3

It should be noted that with 24 terminals the cpu was nearly saturated, with a utilization of over 97%.

The percent errors in the stretch factors, due to passing to periodic models, were:

NUMBER OF TERMINALS	PERIODIC MEAN	PERIODIC MAX	ADJUSTED MEAN	ADJUSTED MAX
12	31.1	66.4	13.4	37.1
16	28.4	72.5	16.8	44.8
24	37.4	88.7	29.6	76.9

The distributions of errors for the 16 terminal workload were:

FOREGROUND-BACKGROUND SCHEDULING AT THE CPU:
ERRORS IN STRETCH FACTORS FOR PERIODIC AND ADJUSTED MODELS

%ERROR	PERIODIC	ADJUSTED
45 -	****	
40 - 45	*	**
35 - 40		
30 - 35	***	*
25 - 30	*	
20 - 25		*
15 - 20	**	****
10 - 15	**	***
5 - 10	**	**
0 - 5	**	****

By spreading the waits w_k farther apart, heavier loads exacerbate the problem of matching an expected wait w_k for a periodic model with the raw model's wait w.

The interaction of the period length with the quantum can be illustrated further by varying both of them together to preserve the quantum usage pattern of each program. The tables below show the small effects on performance resulting from doubling and quadrupling the quantum and the adjusted period lengths of the programs for the 16 terminal workload. (Note that a program's total resource demands limits the possible expansion of its period.) A practical consequence of this period expansion is the corresponding shortening of the running time of the simulator: the running time is approximately inversely proportional to the period expansion, due to the reduction in the number of events to be processed.

EXPANDING THE PERIOD AND THE QUANTUM

Global performance:

INDEX	RAW	2xPERIOD	%ERROR	4xPERIOD	%ERROR
throughput	0.0912	0.0923	+1.2	0.0917	+0.5
cpu queue 0	1.62	1.68	+3.7	1.69	+4.3
disk 1 queue	0.185	0.195	+5.4	0.202	+9.2
disk 2 queue	0.0217	0.0240	+10.6	0.0232	+6.9
disk 3 queue	0.0305	0.0307	+0.7	0.0302	-1.0

Percent errors in stretch factors:

	ADJUSTED	DOUBLED	QUADRUPLED
mean	16.8	16.1	18.7
maximum	44.8	41.5	41.4

Thus, not much accuracy is lost in return for the dramatic saving in running time.

5. Conclusions

Program dynamics had very little influence on the throughput in all cases; if this is the main performance index to be predicted, there is no harm in passing to periodic models and expanding the period lengths to save simulation time. Queue lengths, however, behaved very erratically when job script requests were tampered with, although less so under heavier loads. However, it appears that altering the distribution of the lengths of requests in the presence of a quantum-based service discipline, rather than the dynamic pattern of requests, was the principal cause of error. Similarly, stretch factors were significantly influenced by modifying request distributions in the presence of a quantum-based service discipline; with some care they can be brought under fair control except under heavy loads. If one is going to replace raw models with periodic

ones, little additional error is introduced into the stretch factor predictions by expanding the period, in coordination with the expansion of quanta, if they are used.

Acknowledgements

It is a pleasure to acknowledge the kind support and advice I received from the PROGRES group in general and from Professor Domenico Ferrari and Juan Porcar in particular. In addition, I would like to thank Bill Joy for providing information on the workings of the Berkeley UNIX system on the VAX.

References

[Bassein82] Bassein, R., "The Influence of Program Dynamics on Performance", MS report, 1982, Department of Electrical Engineering and Computer Science, U. C. Berkeley.

[Bryant80] Bryant, R. M., *SIMPAS User Manual*, Computer Sciences Technical Report #391, 1980, University of Wisconsin - Madison.

[Cooper81] Cooper, R. B., *Introduction to Queueing Theory*, 1981, North-Holland Publ. Co., New York.

[Denning77] Denning, P. J. and Buzen, J. P., "Operational analysis of queueing networks", in *Proc. Third Int. Symp. Computer Performance Modeling, Measurement, and Evaluation*, 1977, North-Holland Publ. Co., Amsterdam, The Netherlands.

[Joy80] Joy, W. N., Comments on the performance of UNIX on the VAX, available from the Department of Electrical Engineering and Computer Science, U. C. Berkeley.

[Joy81] Joy, W. N., Conversation.

[Mead78] Mead, R. L. and Schwetman, H. L., "Job scripts - A workload description based on system event data", *National Computer Conference*, 1978, pp. 457-464.

[Serazzi81] Serazzi, G., "The Dynamic Behavior of Computer Systems", *Experimental Computer Performance Evaluation*, 1981, North-Holland Publ. Co., pp.127-163.

Appendix I: The workloads and results

The three workloads consisted of the following 17 programs: as, cat, ccom, cpp, du, ex, grep, ld, ls, mail, more, pc0, pi, ps, px, troff, and w. The number of times each appeared in a simulation may be found in the **COUNT** column of the output of the simulator.

Excerpts from terminal scripts follow; an asterisk preceding a program name indicates that no between-job think time should precede the execution of the program.

A login sequence:

mail w ls more ls

A C editing and execution sequence:

ls ex cpp *ccom ls cat ex ps cpp *ccom *as *ld ls cat ex cpp *ccom *as *ld ls

A Pascal editing and execution sequence:

ls ex pi ls cat ex pi *px ls ps pc0 *as *ld ls cat ex pc0 *as *ld ls

An editing sequence:

ls ex ls ex ls ex ls troff

A logout sequence:

ps ls more ls mail ls du

The output from a sample raw simulation and the corresponding adjusted periodic simulation follows. The workload for the sample simulations had a mean between-job think time of 30 seconds with 16 terminals.

The performance indices reported for each program are the degree of multiprogramming during its execution, the number of context switches, the cpu

stretch factor, and the disk waits. The system indices reported are the interarrival times, service times, utilizations, and queue lengths. The raw simulator prints four lines for each index: the first is the mean over 10 simulations of the mean of the index for each simulation; the second is the standard deviation over 10 simulations of the mean of the index for each simulation; the third is the mean over 10 simulations of the standard deviation of the index for each simulation; the fourth is the standard deviation over 10 simulations of the standard deviation of the index for each simulation. The periodic simulator prints the mean and standard deviation of the index for its single simulation.

Raw Simulation (vaxsim)

Program steps file: prog.st
Program arrival file: 16t301

Memory size = 4.00e+03
Memory reference rate = 1.00e+06
Quantum = 0.250000
Priority decay rate = 0.900000
Top cpu level = 31
Context switch time: 0.00027000
Number of disks = 3; seek, latency, and block I/O times:
0.03000000 0.02500000 0.0012407
0.03000000 0.02600000 0.0012407
0.03000000 0.00833000 0.0008333
Cluster factor = 2.00
20 terminals at 0.0011458 sec. per char.
20 terminals at 0.0366667 sec. per char.
Seed for page fault stream = 123487
Seed for seek time stream = 78653
Seed for latency time stream = 124508

Number of active terminals for termination: 8.000000
Number of simulator runs: 10
Using virtual memory.

Statistics collected from 27.89210

Statistics collected from 27.89210

Statistics collected from 27.89210

Statistics collected from 27.89210

Statistics collected from 27.89210

Statistics collected from 30.19731

Statistics collected from 30.19731

Statistics collected from 30.19731

Statistics collected from 30.19731

Statistics collected from 30.19731

Terminated at 5002.625 (45.995)
Number of jobs = 453.800 (3.490)
Throughput = 0.0912444 (0.0004905)
Normal terminations: 1.000000

PROGRAM	COUNT	MLTPRG	CTXTSW	CPU	DISKS		
as	18.600	12.431	138.002	8.70050715	0.65084294	0.00000000	0.03987800
	(	0.292	1.498	0.88289278	0.13731648	0.00000000	0.00958702)
	0.516	2.033	9.722	5.92414252	0.39727506	0.00000000	0.04334269
	(	0.441	5.494	0.78614938	0.06661262	0.00000000	0.00748242)
ccom	14.900	12.860	239.078	4.56269648	0.14282624	0.00000000	0.13580193
	(	0.093	1.830	0.15420800	0.02760095	0.00000000	0.02137486)
	0.316	2.012	13.965	1.43386135	0.13067263	0.00000000	0.10447464
	(	0.127	4.178	0.18829584	0.03149237	0.00000000	0.02147787)
cpp	15.000	12.369	38.207	3.69048019	0.10288110	0.00000000	0.02188844
	(	0.245	1.079	0.21868336	0.01606335	0.00000000	0.01404576)
	0.000	2.220	9.739	1.41406006	0.10162198	0.00000000	0.03442619
	(	0.283	1.179	0.09222523	0.01985715	0.00000000	0.01726116)
du	11.600	11.902	55.666	2.89804994	0.17862327	0.00000000	0.00000000
	(	0.500	1.565	0.22887401	0.03147474	0.00000000	0.00000000)
	0.699	2.143	11.341	1.50320703	0.14581311	0.00000000	0.00000000
	(	0.574	1.170	0.18249828	0.04012018	0.00000000	0.00000000)
ex	44.000	13.234	308.955	7.54421468	0.65953737	0.00184932	0.00000000
	(	0.016	0.704	0.15735729	0.03225048	0.00129991	0.00000000)
	0.000	0.747	9.829	1.95667847	0.20401533	0.00864219	0.00000000
	(	0.024	0.642	0.10224622	0.02039962	0.00543288	0.00000000)

find	Ø.ØØØ	Ø.ØØØ	Ø.ØØØ	Ø.ØØØØØØØØ	Ø.ØØØØØØØØ	Ø.ØØØØØØØØ	Ø.ØØØØØØØØ
	(Ø.ØØØ	Ø.ØØØ	Ø.ØØØ	Ø.ØØØØØØØØ	Ø.ØØØØØØØØ	Ø.ØØØØØØØØ	Ø.ØØØØØØØØ)
	Ø.ØØØ	Ø.ØØØ	Ø.ØØØ	Ø.ØØØØØØØØ	Ø.ØØØØØØØØ	Ø.ØØØØØØØØ	Ø.ØØØØØØØØ
	(Ø.ØØØ	Ø.ØØØ	Ø.ØØØ	Ø.ØØØØØØØØ	Ø.ØØØØØØØØ	Ø.ØØØØØØØØ	Ø.ØØØØØØØØ)
grep	15.3ØØ	11.648	36.183	4.32915142	Ø.19749257	Ø.ØØ68Ø341	Ø.ØØØØØØØØ
	(Ø.35Ø	Ø.5Ø5	Ø.5Ø5	Ø.32526Ø45	Ø.Ø2629979	Ø.ØØ49Ø872	Ø.ØØØØØØØØ)
	Ø.ØØØ	1.96Ø	3.ØØ7	1.65726Ø36	Ø.Ø8375Ø9Ø	Ø.Ø1921Ø26	Ø.ØØØØØØØØ
	(Ø.454	1.194	Ø.5Ø3	Ø.5Ø313255	Ø.Ø2583266	Ø.Ø1138745	Ø.ØØØØØØØØ)
ld	18.4ØØ	11.644	25.775	5.91333691	Ø.17484155	Ø.ØØØØØØØØ	Ø.ØØØØØØØØ
	(Ø.312	Ø.623	Ø.623	1.44726485	Ø.Ø2923Ø94	Ø.ØØØØØØØØ	Ø.ØØØØØØØØ)
	Ø.516	2.3Ø3	2.589	5.96293Ø13	Ø.13114442	Ø.ØØØØØØØØ	Ø.ØØØØØØØØ
	(Ø.293	1.2Ø7	1.2Ø7	3.8Ø517381	Ø.Ø3375236	Ø.ØØØØØØØØ	Ø.ØØØØØØØØ)
ls	134.6ØØ	12.335	94.314	3.16877805	Ø.Ø6956781	Ø.ØØ183963	Ø.ØØØØØØØØ
	(Ø.Ø72	Ø.577	Ø.577	Ø.Ø4891643	Ø.ØØ3ØØ762	Ø.ØØ92252	Ø.ØØØØØØØØ)
	1.776	1.998	9.511	1.15912714	Ø.Ø6971213	Ø.ØØ958276	Ø.ØØØØØØØØ
	(Ø.183	Ø.946	Ø.946	Ø.1ØØ47913	Ø.ØØ66Ø982	Ø.ØØ317755	Ø.ØØØØØØØØ)
mail	28.4ØØ	11.769	3Ø.464	8.43833751	Ø.ØØØØØØØØ	Ø.ØØ185Ø36	Ø.ØØØØØØØØ
	(Ø.356	Ø.141	Ø.141	Ø.23Ø2Ø581	Ø.ØØØØØØØØ	Ø.ØØ148357	Ø.ØØØØØØØØ)
	Ø.516	1.694	1.Ø64	3.Ø3626Ø34	Ø.ØØØØØØØØ	Ø.ØØ823136	Ø.ØØØØØØØØ
	(Ø.18Ø	Ø.122	Ø.122	Ø.25471Ø96	Ø.ØØØØØØØØ	Ø.ØØ686627	Ø.ØØØØØØØØ)
more	44.ØØØ	11.729	52.778	8.236Ø5845	Ø.14Ø1Ø695	Ø.ØØ368857	Ø.ØØØØØØØØ
	(Ø.Ø98	Ø.417	Ø.417	Ø.26541478	Ø.Ø1659193	Ø.ØØ239194	Ø.ØØØØØØØØ)
	Ø.816	1.847	3.738	3.99441ØØ9	Ø.Ø9855960	Ø.Ø1336818	Ø.ØØØØØØØØ
	(Ø.168	Ø.383	Ø.383	Ø.35Ø83392	Ø.Ø1483Ø16	Ø.ØØ678936	Ø.ØØØØØØØØ)
pcØ	9.ØØØ	12.927	258.911	3.85626992	Ø.28Ø36119	Ø.19344992	Ø.Ø62Ø751Ø
	(Ø.227	6.435	6.435	Ø.43947Ø15	Ø.Ø49Ø6588	Ø.Ø864Ø514	Ø.Ø2519729)
	Ø.ØØØ	1.251	25.847	1.64Ø957Ø4	Ø.13275997	Ø.22413415	Ø.Ø6Ø29954
	(Ø.396	11.626	11.626	Ø.8515Ø746	Ø.Ø3576349	Ø.12945326	Ø.Ø2416193)
pi	1Ø.ØØØ	13.926	19Ø.66Ø	3.64874356	Ø.37398445	Ø.Ø6634834	Ø.ØØØØØØØØ
	(Ø.17Ø	3.846	3.846	Ø.25871887	Ø.Ø6873563	Ø.Ø3453482	Ø.ØØØØØØØØ)
	Ø.ØØØ	Ø.863	37.546	1.33354125	Ø.25158261	Ø.11293889	Ø.ØØØØØØØØ
	(Ø.139	4.495	4.495	Ø.21578927	Ø.Ø6628594	Ø.Ø7254481	Ø.ØØØØØØØØ)
ps	23.3ØØ	12.52Ø	26.911	4.21931968	Ø.Ø914Ø518	Ø.ØØØØØØØØ	Ø.ØØØØØØØØ
	(Ø.281	Ø.888	Ø.888	Ø.34254763	Ø.Ø2Ø65ØØ8	Ø.ØØØØØØØØ	Ø.ØØØØØØØØ)
	Ø.675	2.Ø54	3.87Ø	3.24237533	Ø.Ø8182Ø4	Ø.ØØØØØØØØ	Ø.ØØØØØØØØ
	(Ø.254	1.Ø16	1.Ø16	Ø.36716598	Ø.Ø1648181	Ø.ØØØØØØØØ	Ø.ØØØØØØØØ)
px	5.ØØØ	13.131	1Ø6.52Ø	1Ø.Ø8763814	Ø.Ø8479931	Ø.ØØ693766	Ø.ØØØØØØØØ
	(Ø.161	1.726	1.726	1.14938165	Ø.Ø2641681	Ø.Ø1877Ø85	Ø.Ø1ØØØ3Ø
	Ø.ØØØ	Ø.641	3.122	4.88947281	Ø.Ø7Ø27Ø64	Ø.Ø129ØØØ4	Ø.ØØØØØØØØ
	(Ø.241	1.319	1.319	Ø.72268Ø99	Ø.Ø3548693	Ø.Ø1414Ø989	Ø.ØØØØØØØØ)
troff	5.ØØØ	12.Ø8Ø	363.38Ø	7.5Ø738295	Ø.4Ø446837	Ø.Ø1368678	Ø.Ø629Ø3Ø7
	(Ø.Ø91	2.388	2.388	Ø.3Ø615Ø51	Ø.Ø5982265	Ø.Ø1453619	Ø.Ø2171284)
	Ø.ØØØ	1.439	18.478	4.27699286	Ø.13569326	Ø.Ø2525457	Ø.Ø6Ø6Ø569
	(Ø.121	4.448	4.448	Ø.34663151	Ø.Ø6761483	Ø.Ø3Ø98566	Ø.Ø24Ø9481)
vaxima	Ø.ØØØ	Ø.ØØØ	Ø.ØØØ	Ø.ØØØØØØØØ	Ø.ØØØØØØØØ	Ø.ØØØØØØØØ	Ø.ØØØØØØØØ
	(Ø.ØØØ	Ø.ØØØ	Ø.ØØØ	Ø.ØØØØØØØØ	Ø.ØØØØØØØØ	Ø.ØØØØØØØØ	Ø.ØØØØØØØØ)
	Ø.ØØØ	Ø.ØØØ	Ø.ØØØ	Ø.ØØØØØØØØ	Ø.ØØØØØØØØ	Ø.ØØØØØØØØ	Ø.ØØØØØØØØ
	(Ø.ØØØ	Ø.ØØØ	Ø.ØØØ	Ø.ØØØØØØØØ	Ø.ØØØØØØØØ	Ø.ØØØØØØØØ	Ø.ØØØØØØØØ)
w	15.3ØØ	11.736	44.685	8.Ø9867359	Ø.Ø6341862	Ø.ØØ222412	Ø.ØØØØØØØØ
	(Ø.178	Ø.88Ø	Ø.88Ø	Ø.73431Ø95	Ø.Ø1424219	Ø.ØØ178849	Ø.ØØØØØØØØ)
	Ø.483	1.593	7.354	7.57552286	Ø.Ø5864Ø195	Ø.ØØ795255	Ø.ØØØØØØØØ
	(Ø.22Ø	1.231	1.231	1.Ø8Ø954Ø8	Ø.Ø1Ø761Ø2	Ø.ØØ637Ø3Ø	Ø.ØØØØØØØØ)
cat	36.ØØØ	12.315	175.145	3.29117278	Ø.78128783	Ø.ØØØØØØØØ	Ø.ØØØØØØØØ
	(Ø.244	2.259	2.259	Ø.13778538	Ø.Ø5542485	Ø.ØØØØØØØØ	Ø.ØØØØØØØØ)
	Ø.471	2.Ø22	23.853	1.Ø2924Ø61	Ø.4Ø2742Ø1	Ø.ØØØØØØØØ	Ø.ØØØØØØØØ
	(Ø.2Ø4	3.764	3.764	Ø.17Ø547Ø8	Ø.Ø6388465	Ø.ØØØØØØØØ	Ø.ØØØØØØØØ)

Mean memory demand: 3.Ø14e+Ø3 (9.Ø99e+Ø2)
Context switch overhead: 14.14341ØØØ (Ø.Ø9686384)

SERVER	ARRIVALS	SERVICE	UTILIZATION	QLENGTH
cpu	Ø.Ø9138212	Ø.ØØØ85317	Ø.8876Ø979 (Ø)	1.61522868

```
      (      Ø.ØØØ40382    Ø.ØØØØ73ØØ Ø.ØØØ3[illegible]    [illegible]3Ø )
             Ø.Ø8561877    Ø.Ø7819841 Ø.31579Ø62    1.[illegible]9151
      (      Ø.ØØ124738    Ø.Ø[illegible]9465 Ø.Ø5477[illegible]    [illegible]85Ø5 )

disk[1]      2.25Ø88383    Ø.Ø4232554 Ø.162844[illegible]    Ø.[illegible]5876
      (      Ø.ØØ173924    Ø.ØØØ13244 Ø.ØØ148399    Ø.ØØ236979 )
             Ø.38152268    Ø.Ø2255ØØ7 Ø.36922623    Ø.[illegible]83451
      (      Ø.Ø1Ø68255    Ø.ØØØ47122 Ø.ØØ136478    [illegible]7886 )

disk[2]      1.78Ø92199    Ø.Ø3Ø59633 Ø.Ø2161665    Ø.[illegible]8200
      (      Ø.Ø1516875    Ø.ØØØ31729 Ø.ØØØ23978    Ø.[illegible]1909 )
             5.22825701    Ø.Ø2154696 Ø.14543884    Ø.[illegible]2935
      (      Ø.15882Ø28    Ø.ØØØ217Ø7 Ø.ØØØ95439    Ø.ØØ13Ø457 )

disk[3]      1.32953159    Ø.ØØ969105 Ø.Ø297ØØ84    Ø.[illegible]7697
      (      Ø.ØØ91Ø995    Ø.ØØØ28795 Ø.ØØØ32158    Ø.ØØØ32823 )
            17.6Ø32Ø889    Ø.Ø2172119 Ø.16977577    Ø.[illegible]9390
      (      Ø.22968228    Ø.ØØØ2Ø6Ø7 Ø.ØØØ89273    Ø.ØØ129836 )
```

Periodic Simulation (perfsim)

Program steps file: prog.pr
Program arrival file: 16t30icre

Memory size = 4.00e+03
Memory reference rate = 1.00e+06
Quantum = 0.250000
Priority decay rate = 0.900000
Top cpu level = 31
Context switch time: 0.0002/000
Number of disks = 3; seek, latency, and block I/O times:
0.030000 0.005000 0.0012407
0.030000 0.006000 0.0012407
0.030000 0.003000 0.0008333
Cluster factor = 2.00
20 terminals at 0.0011458 sec. per char.
20 terminals at 0.0366667 sec. per char.

Number of active terminals for termination: 8.00000
Using virtual memory.

Statistics collected from 30.19731
Terminated at 4917.212
Number of jobs processed = 447
Mean throughput = 0.0914669
Normal termination.

PROGRAM	COUNT	MLTPRG	CTXTSW	CPU	DISKS		
as	18	12.795	95.944	8.83786341	1.95573591	0.00000000	0.09053098
		1.830	0.236	3.41675917	1.12429607	0.00000000	0.11178580
ccom	14	13.262	205.714	5.47595979	0.58220509	0.00000000	0.01852409
		1.375	26.205	3.52883051	0.59671174	0.00000000	0.04333016
cpp	15	12.821	36.400	4.38078278	0.13883735	0.00000000	0.02060336
		1.768	6.801	1.75934801	0.12136779	0.00000000	0.02912916
du	13	11.948	55.308	2.98741435	1.02215988	0.00000000	0.00000000
		2.020	3.545	1.21368941	0.65174802	0.00000000	0.00000000
ex	44	13.212	281.091	7.70025466	0.80713311	0.15422752	0.00000000
		0.728	26.328	2.34736268	0.23775985	0.12530758	0.00000000
find	0	0.000	0.000	0.00000000	0.00000000	0.00000000	0.00000000
		0.000	0.000	0.00000000	0.00000000	0.00000000	0.00000000
grep	15	11.211	26.933	3.83239458	0.02371066	0.00331007	0.00000000
		2.099	0.258	1.17125093	0.07058033	0.00642701	0.00000000
ld	18	12.286	20.000	7.98028820	0.51635885	0.00000000	0.00000000
		1.594	0.000	4.49987917	0.51826066	0.00000000	0.00000000
ls	132	12.363	87.114	2.98274879	0.02508651	0.01553545	0.00000000
		1.847	13.595	1.14587057	0.06088276	0.03861513	0.00000000
mail	28	11.807	26.786	9.87721494	0.00000000	0.01352974	0.00000000
		1.831	4.779	5.37787482	0.00000000	0.03336894	0.00000000
more	41	12.150	45.951	6.97947307	0.09396498	0.01944844	0.00000000
		1.589	4.653	2.70390498	0.09252020	0.04947875	0.00000000
pc0	9	13.294	140.111	2.12718879	0.00535373	0.03629873	0.00000000
		1.543	2.667	0.24355017	0.01606118	0.08777224	0.00000000
pi	10	14.098	87.000	4.32909950	0.04620528	0.01615820	0.00000000
		1.140	7.196	1.76280525	0.09308430	0.03619294	0.00000000
ps	21	13.240	19.667	2.38684108	0.10727215	0.00000000	0.00000000
		1.608	3.039	0.89472431	0.10693919	0.00000000	0.00000000
px	5	13.091	90.400	7.62846524	0.00443198	0.00000000	0.00000000
		0.501	7.893	3.59629972	0.00683310	0.00000000	0.00000000
troff	5	12.249	333.200	7.43912267	0.37693419	0.02996297	0.03371691
		1.482	34.967	4.01900854	0.35679363	0.04008710	0.06488504

vaxima	0	0.000	0.000	0.00000000	0.00000000	0.00000000	0.00000000
		0.000	0.000	0.00000000	0.00000000	0.00000000	0.00000000
w	15	11.884	23.333	6.94994132	3.81129695	0.00000000	0.00000000
		1.882	1.339	5.85247561	0.32508891	0.00000000	0.00000000
cat	36	12.069	163.278	2.97543971	1.32887701	0.00000000	0.00000000
		2.465	24.956	0.82493873	1.14259742	0.00000000	0.00000000
q1test	0	0.000	0.000	0.00000000	0.00000000	0.00000000	0.00000000
		0.000	0.000	0.00000000	0.00000000	0.00000000	0.00000000
q2test	0	0.000	0.000	0.00000000	0.00000000	0.00000000	0.00000000
		0.000	0.000	0.00000000	0.00000000	0.00000000	0.00000000
q3test	0	0.000	0.000	0.00000000	0.00000000	0.00000000	0.00000000
		0.000	0.000	0.00000000	0.00000000	0.00000000	0.00000000
q4test	0	0.000	0.000	0.00000000	0.00000000	0.00000000	0.00000000
		0.000	0.000	0.00000000	0.00000000	0.00000000	0.00000000

Mean memory demand: 3.058e+03 (8.713e+02)
Context switch overhead: 12.02445000

SERVER	ARRIVALS	SERVICE	UTILIZATION	QLENGTH
cpu	0.10240971	0.09107760	0.89170379 (0)	1.71562510
	0.12019659	0.06658099	0.31075578	1.79843447
disk[1]	0.19270929	0.03178358	0.16492682	0.19961308
	0.23766933	0.03816138	0.37111998	0.49680066
disk[2]	0.14949033	0.00327577	0.02191278	0.02333182
	0.23674500	0.01014770	0.14639995	0.15636355
disk[3]	2.52533421	0.07549826	0.02989333	0.03019357
	24.32926353	0.02084172	0.17031573	0.17393206

Appendix II: The computation of waits in a foreground-background queue

As defined in the text of the report, w_k is the expected wait experienced by a request of more than k-1 quanta but not more than k quanta; call such a request a "k-request". Let q be the quantum. The w_k may be estimated as follows. Assume that interarrival times are exponentially distributed with mean λ^{-1} and that service times, for the workload as a whole, are exponentially distributed with mean μ^{-1}. As a result of the foreground-background service discipline, a k-request waits for, essentially, at most the first k quanta of any other request in the queue during the busy period in which the k-request arrives: a newly arriving k-request will complete without waiting for the portion exceeding k quanta of preceding requests and will have to wait for the first k-1 quanta of any request entering before it completes its service. (Actually, this is exactly correct only if an incoming job pre-empts one in service and there is no overhead for context switches.) Now let W_i be the expected wait experienced by all requests not exceeding i quanta, that is, 1-requests through i-requests. W_i can be computed by the Pollaczeck-Khintchine formula [Cooper81] for the M/G/1 queue with service time frequency function $s_i(t)$ equal to $\mu e^{-\mu t}$ for t = 0 to iq, having a mass of $e^{-i\mu q}$ at t = iq, and vanishing beyond iq. Thus, $W_i = \frac{\rho_i \tau_i}{2(1-\rho_i)}\left[1+\frac{\sigma_i^2}{\tau_i^2}\right]$, where τ_i and σ_i^2 are the mean and variance, respectively, of service times based on the service time frequency function $s_i(t)$, and $\rho_i = \lambda\tau_i$. So $\tau_i = \int_0^{iq} t\,\mu e^{-\mu t}\,dt + iqe^{-i\mu q} = \mu^{-1}(1-e^{-i\mu q})$ and $\sigma_i^2 = \int_0^{iq} t^2\mu e^{-\mu t}\,dt + (iq)^2 e^{-i\mu q} - \tau_i^2 = \mu^{-2}(1-2i\mu q e^{-i\mu q} - e^{-2i\mu q})$. Finally, $W_i = \frac{\rho\tau[1-e^{-i\mu q}(1+i\mu q)]}{1-\rho(1-e^{-i\mu q})}$, where $\rho = \lambda\tau$ and $\tau = \mu^{-1}$. If $S(t) = 1-e^{-\mu t}$ for all t, the actual distribution function of service requests, then the fraction of requests using at most i quanta which actually use no more than i-1 quanta is

$f_i = \frac{S((i-1)\mu q)}{S(i\mu q)}$. The w_k can be computed from the W_i by $w_k = \frac{W_k - f_k W_{k-1}}{1-f_k}$. For q = 0.25 seconds, λ^{-1} = 0.09, and μ^{-1} = 0.08 (approximate values from the 16 terminal raw simulation), the values of w_k for k = 1 to 4 are: 0.39, 5.94, 10.13, 12.30. (By inserting test jobs with fixed request lengths, I measured these values as: 0.31, 4.47, 15.42, and 17.76, respectively, close enough for the purpose here.)

Given the w_k and the distribution of request lengths for a particular program, its expected wait and the necessary adjustment of period length can be estimated as described before. To keep the periodic program models independent of the system model and to allow the raw models to be disposed of, this should ideally be accomplished from the periodic job scripts, without recourse to the raw job scripts. However, the importance of the exact relationship of cpu request lengths to the size of the quantum makes this impossible. For example, one could assume an exponential distribution of requests from a particular program and compute its expected wait from its mean request. The outliers in the following table of request distributions show why such an assumption is unsatisfactory:

PROGRAM	NUMBER OF REQUESTS USING 1 TO 8+ QUANTA								
as	85	7	0	0	0	0	1	0	
ccom	46	23	27	10	4	0	0	0	
cpp	41	0	0	0	0	0	0	0	
du	56	3	0	0	0	0	0	0	
ex	314	2	0	0	0	0	0	0	
grep	28	1	0	0	0	0	0	0	
ld	16	2	0	1	0	0	0	0	
ls	90	3	0	0	0	0	0	0	
mail	31	0	0	0	0	0	0	0	
more	47	2	0	0	0	0	0	0	
pc0	107	31	3	0	0	0	0	0	
pi	63	24	4	0	0	0	0	0	
ps	17	4	0	0	0	0	0	0	
px	12	1	1	2	2	0	0	0	... 1
troff	94	28	29	21	6	1	0	0	
w	36	2	0	0	0	1	0	0	
cat	173	6	0	0	0	0	0	0	

Based on the above table and the computation of the w_k, I adjusted the period lengths by the following factors:

PROGRAM	ADJUSTMENT FACTOR
as	2.0
ccom	1.0
cpp	1.0
du	1.0
ex	1.0
grep	1.0
ld	2.0
ls	1.0
mail	1.0
more	1.0
pc0	1.0
pi	2.0
ps	1.0
px	0.75
troff	1.0
w	3.0
cat	1.0